Combining typing and size constraints for checking the termination of higher-order conditional rewrite systems

Frédéric Blanqui (INRIA) and Colin Riba (INPL)

LORIA*, Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France

Abstract. In a previous work, the first author extended to higher-order rewriting and dependent types the use of size annotations in types, a termination proof technique called type or size based termination and initially developed for ML-like programs. Here, we go one step further by considering conditional rewriting and explicit quantifications and constraints on size annotations. This allows to describe more precisely how the size of the output of a function depends on the size of its inputs. Hence, we can check the termination of more functions. We first give a general type-checking algorithm based on constraint solving. Then, we give a termination criterion with constraints in Presburger arithmetic. To our knowledge, this is the first termination criterion for higher-order conditional rewriting taking into account the conditions in termination.

1 Introduction

We are interested in automatically checking the termination of the combination of β-reduction and higher-order conditional rewrite rules. There are two important approaches to higher-order rewriting: rewriting on $\beta\overline{\eta}$-normal forms [17], and the combination of β-reduction and term rewriting [16]. The relation between both has been studied in [20]. The second approach is more atomic since a rewrite step in the first approach can be directly encoded by a rewrite step together with β-steps in the second approach. In this paper, we consider the second approach, restricted to first-order pattern-matching (we do not allow abstractions in rule left-hand side). Following [7], our results could perhaps be extended to higher-order pattern-matching.

The combination of β-reduction and rewriting is naturally used in proof assistants implementing the proposition-as-type and proof-as-object paradigm. In these systems, two propositions equivalent modulo β-reduction and rewriting are identified (*e.g.* $P(2+2)$ and $P(4)$). This is essential for enabling users to formalize large proofs with many computations, as recently shown by Gonthier and Werner's proof of the Four Color Theorem in the Coq proof assistant. However, for the system to be able to check the correctness of user proofs, it must

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP

at least be able to check the equivalence of two terms. Hence, the necessity to have termination criteria for the combination of β-reduction and rewriting.

In Coq, rewriting is restricted to the reductions associated to inductive types like in functional programming languages with pattern-matching. Such reductions correspond to constructor-based rewriting. This is the kind of rewrite systems we are going to consider in this paper. A more general form of rewriting is studied in [9, 6] (matching on defined symbols and matching modulo).

Currently, Coq accepts only functions in the definition of which recursive calls are made on arguments that are structurally smaller. For first-order functions, this corresponds to restrict rewrite systems to simply terminating ones, that is, to the ones that can be proved terminating by an ordering containing the subterm relation. However, many interesting systems are not simply terminating. Consider for instance the following definition of division on natural numbers:

$$\begin{array}{rcl} \mathsf{minus}\,\mathsf{0}\,x & \to & \mathsf{0} \\ \mathsf{minus}\,x\,\mathsf{0} & \to & x \\ \mathsf{minus}\,(\mathsf{s}\,x)\,(\mathsf{s}\,y) & \to & \mathsf{minus}\,x\,y \\ \mathsf{div}\,\mathsf{0}\;y & \to & \mathsf{0} \\ \mathsf{div}\,(\mathsf{s}\,x)\,y & \to & \mathsf{s}\,(\mathsf{div}\,(\mathsf{minus}\,x\,y)\,y) \end{array}$$

Considering that minus is applied to strongly normalizing arguments and that the *size* of a term is the height of its normal form, one can easily prove, by induction on the size of t, that the size of $v = (\mathsf{minus}\,t\,u)$ is less than or equal to the size of t, hence that this definition of minus terminates:

- If v matches the first rule, then $t = \mathsf{0}$ and the normal form of v, which is $\mathsf{0}$, has the same size as t.
- If v matches the second rule, then v has the same normal form as t.
- If v matches the third rule, then $t = \mathsf{s}t'$, $u = \mathsf{s}u'$ and, by induction hypothesis, the normal form of v has a size smaller than t', hence smaller than t.

The idea of size or type based termination, initiated in [15] and developed by various authors for ML-like definitions [11, 22, 1–4] and rewriting and dependent types [8, 5], consists in extending the underlying type system by replacing a base type B by an infinite family of base types $(\mathsf{B}^{\mathfrak{a}})_{\mathfrak{a}\in\mathbb{N}}$, a term of type $\mathsf{B}^{\mathfrak{a}}$ being by construction of size smaller than or equal to $\mathfrak{a}$ (except in [22], see later). Then, for ensuring termination, one can restrict in function definitions recursive calls to arguments whose size, by typing, is smaller.

For instance, in all these systems, one can easily (type-)check that minus has for type something similar to $\forall\alpha\beta\mathsf{N}^{\alpha} \Rightarrow \mathsf{N}^{\beta} \Rightarrow \mathsf{N}^{\alpha}$. Hence, assuming that $x : \mathsf{N}^{\alpha}$ and $y : \mathsf{N}^{\beta}$, one can easily (type-)check that $\mathsf{minus}\,x\,y : \mathsf{N}^{\alpha}$ while $\mathsf{s}x : \mathsf{N}^{\alpha+1}$. Thus, the recursive call to div in the last rule can be allowed.

Note that higher-order inductive types, *i.e.* types having constructors with recursive arguments of higher-order type, require families indexed by ordinals. In the present paper, we restrict our attention to first-order inductive types since higher-order inductive types have already been studied in previous works. Note also that interpreting $\mathsf{B}^{\mathfrak{a}}$ by the set of terms of size smaller than or equal to $\mathfrak{a}$ requires subtyping since $t : \mathsf{B}^{\mathfrak{b}}$ whenever $t : \mathsf{B}^{\mathfrak{a}}$ and $\mathfrak{a} \leq \mathfrak{b}$.

However, without explicit existential quantifications and constraints over size annotations, one cannot (type-)check that the following function has type $\mathsf{N} \Rightarrow \forall\alpha\mathsf{L}^{\alpha} \Rightarrow \exists\beta\gamma(\alpha = \beta + \gamma)\mathsf{L}^{\beta} \times \mathsf{L}^{\gamma}$:

$$\begin{array}{rcl} \mathsf{pivot}\, x\, \mathsf{nil} & \to & (\mathsf{nil}\,, \mathsf{nil}\,) \\ \mathsf{pivot}\, x\, (\mathsf{cons}\, y\, l) & \to & \mathsf{let}\, z = \mathsf{pivot}\, x\, l\, \mathsf{in} \\ & & \mathsf{if}\, (\mathsf{le}\, y\, x)\, \mathsf{then}\, (\mathsf{cons}\, y\, (\mathsf{fst}\, z), \mathsf{snd}\, z) \\ & & \mathsf{else}\, (\mathsf{fst}\, z, \mathsf{cons}\, y\, (\mathsf{snd}\, z)) \end{array}$$

Such a type is necessary for proving that some sorting functions are size preserving, *i.e.* have type $\forall\alpha\mathsf{L}^{\alpha} \Rightarrow \mathsf{L}^{\alpha}$. To the best of our knowledge, only Xi considers such explicit quantifications and constraints [22]. In this work, $\mathsf{B}^{\mathfrak{a}}$ is interpreted as the set of terms of size $\mathfrak{a}$. Note that, with this interpretation, the type of terms of size smaller than $\mathfrak{a}$ can be represented by $\exists\alpha(\alpha \leq \mathfrak{a})\mathsf{B}^{\alpha}$. However, we cannot apply Xi's results on the problem we are interested in for the following reasons:

- Xi considers ML-like function definitions based on `letrec`/`match` constructions while we are interested in definitions based on rewrite rules.
- Xi is interested in the termination of closed terms with call-by-value evaluation strategy while we are interested in the strong normalization of open terms.
- Xi has a two-level approach. He considers an intermediate system where not only types but also terms are annotated by size informations, and proves that terms typable in this system are terminating. Then, for proving the termination of an unannotated term, he must infer the necessary size annotations, which may not be possible. This elaboration process is described in [21].

In the present paper, we extend the simply typed part of [8] with conditional rewriting and explicit quantifications and constraints over size annotations, without using an intermediate system. As Xi and in contrast with [8], we do not consider higher-order inductive types and interpret $\mathsf{B}^{\mathfrak{a}}$ as the set of terms of size $\mathfrak{a}$. The integration of both works should not create too much difficulties. Hence, we get a powerful termination criterion for the combination of β-reduction and higher-order conditional rewriting, based on type-checking and constraint solving. To our knowledge, this is the first termination criterion for higher-order conditional rewriting taking into account the conditions in termination.

In Section 2, we define a system with constrained types. In Section 3, we give a general type-checking algorithm based on constraint solving. In Section 4, we present a general termination proof technique based on Tait's method for proving the termination of β-reduction. In Section 5, we give a termination criterion based on type-checking with constraints in Presburger arithmetic.

2 A system with constrained types

Terms. The set $\mathcal{T}$ of *terms* is inductively defined as follows:

$$t \in \mathcal{T} ::= x \mid \mathsf{c} \mid \mathsf{f} \mid \lambda x t \mid tt \mid (t,t) \mid \mathsf{fst}\, t \mid \mathsf{snd}\, t \mid \mathsf{let}\, x = t\, \mathsf{in}\, t \mid \mathsf{if}\, t\, \mathsf{then}\, t\, \mathsf{else}\, t$$

where $x \in \mathcal{X}$ is a term variable, $\mathsf{c} \in \mathcal{C}$ is a *constructor* symbol and $\mathsf{f} \in \mathcal{F}$ is a *function symbol*. We assume that $\mathcal{C}$ contains true and false. As usual, terms are considered up to renaming of bound variables. By $\boldsymbol{t}$, we denote a sequence of terms $t_1, \ldots, t_n$ of length $|\boldsymbol{t}| = n \geq 0$. Term substitutions are denoted by $\sigma, \theta, \ldots$ or their explicit mappings $\binom{\boldsymbol{t}}{\boldsymbol{x}}$. By $\sigma + \theta$, we denote the substitution equal to θ on $\mathrm{dom}(\theta)$ and to σ on $\mathrm{dom}(\sigma) \setminus \mathrm{dom}(\theta)$. The set $\mathcal{P}$ of (constructor) *patterns* is inductively defined by $p \in \mathcal{P} ::= x \mid \mathsf{c}\,\boldsymbol{p}$.

Size annotations. Let $\mathcal{S} = \{nat, bool\}$ be the set of *size sorts*. We assume given a $\mathcal{S}$-sorted first-order term algebra $\mathcal{A}$ for *size expressions* $a, b, \ldots$ whose variables are denoted by $\alpha, \beta, \ldots$ We assume that $\mathcal{A}$ at least contains the symbols $0 : nat$, $1 : nat$, $+ : nat \times nat \Rightarrow nat$, $max : nat \times nat \Rightarrow nat$, $\mathrm{t} : bool$ and $\mathrm{f} : bool$. For each sort s, we assume given a well-founded interpretation domain $(\mathcal{D}_s, >_{\mathcal{D}_s})$. For $bool$, we take $\mathcal{D}_{bool} = \{\mathfrak{t}, \mathfrak{f}\}$. In the following, let $\mathsf{true}^* = \mathrm{t}$ and $\mathsf{false}^* = \mathrm{f}$; $\mathrm{t}^* = \mathfrak{t}$ and $\mathrm{f}^* = \mathfrak{f}$; $\mathfrak{t}^* = \mathsf{true}$ and $\mathfrak{f}^* = \mathsf{false}$. Elements of $\mathcal{D}_s$ are denoted by $\mathfrak{a}, \mathfrak{b}, \ldots$ Valuations are denoted by $\mu, \nu, \ldots$ Size substitutions are denoted by $\varphi, \psi, \ldots$

Constraints. Let a *constraint* be a first-order formula over $\mathcal{A}$, $\mathbb{C}$ be a class of constraints containing $\top$ and $\mathrm{FV}(C)$ be the variables free in C. We denote by $\mu \models C$ the fact that a valuation μ satisfies C; by $\vdash C$ the fact that, for all valuation μ such that $\mathrm{FV}(C) \subseteq \mathrm{dom}(\mu)$, $\mu \models C$, and by $C \equiv D$ the fact that $\vdash C \Leftrightarrow D$. We consider constraints up to the logical equivalence $\equiv$.

Types. We assume given a set $\mathcal{B}$ of type names containing bool. Let $\kappa_{\mathsf{bool}} = bool$ and, for all $\mathsf{B} \neq \mathsf{bool}$, $\kappa_{\mathsf{B}} = nat$ (except bool that is annotated by booleans, types are annotated by natural numbers). Types are defined as follows:

$$\begin{array}{rl}
\text{types } T \in \mathbb{T} ::= & \mathsf{B}^a \mid T \Rightarrow T \mid T \times T \mid \forall \boldsymbol{\alpha} P T \mid \exists \boldsymbol{\alpha} P T \\
\text{simple types } S \in \mathbb{S} ::= & \exists \alpha \mathsf{B}^\alpha \mid S \Rightarrow S \mid S \times S \\
\text{basic types } B \in \mathbb{B} ::= & \mathsf{B}^a \mid B \times B \\
\exists\text{-basic types } E \in \mathbb{E} ::= & B \mid \exists \boldsymbol{\alpha} P E \quad \text{with } \vdash \exists \boldsymbol{\alpha} P
\end{array}$$

where $\mathsf{B} \in \mathcal{B}$ is a type name, $a \in \mathcal{A}$ is a size expression of sort κ_{B} and $P \in \mathbb{C}$ is a constraint. In the following, we use the following abbreviations: $\forall \boldsymbol{\alpha} T = \forall \boldsymbol{\alpha} \top T$ and $\mathsf{B} = \exists \alpha \mathsf{B}^\alpha$. There is a natural transformation from $\mathbb{T}$ to $\mathbb{S}$: let $\overline{\mathsf{B}^a} = \exists \alpha \mathsf{B}^\alpha$, $\overline{\exists \alpha P T} = \overline{\forall \alpha P T} = \overline{T}$, $\overline{T \Rightarrow U} = \overline{T} \Rightarrow \overline{U}$ and $\overline{T \times U} = \overline{T} \times \overline{U}$.

Subtyping. We define a constraint-based subtyping relation. Let $C \vdash T \leq U$ iff $\vdash C \supset (\!|T \leq U|\!)$ where $(\!|T \leq U|\!)$ is inductively defined as follows:

- $(\!|B^a \leq B^b|\!) = (a = b)$
- $(\!|T \Rightarrow U \leq T' \Rightarrow U'|\!) = (\!|T' \leq T|\!) \wedge (\!|U \leq U'|\!)$
- $(\!|T \times U \leq T' \times U'|\!) = (\!|T \leq T'|\!) \wedge (\!|U \leq U'|\!)$
- $(\!|T \leq \exists \boldsymbol{\alpha} P U|\!) = \exists \boldsymbol{\alpha}(P \wedge (\!|T \leq U|\!))$ $(\boldsymbol{\alpha} \notin T,\ T \neq \exists \beta Q V)$
- $(\!|\exists \boldsymbol{\alpha} P U \leq T|\!) = \forall \boldsymbol{\alpha}(P \supset (\!|U \leq T|\!))$ $(\boldsymbol{\alpha} \notin T)$
- $(\!|T \leq \forall \boldsymbol{\alpha} P U|\!) = \forall \boldsymbol{\alpha}(P \supset (\!|T \leq U|\!))$ $(\boldsymbol{\alpha} \notin T)$
- $(\!|\forall \boldsymbol{\alpha} P U \leq T|\!) = \exists \boldsymbol{\alpha}(P \wedge (\!|U \leq T|\!))$ $(\boldsymbol{\alpha} \notin T,\ T \neq \forall \beta Q V)$

Typing. An *environment* is a finite mapping Γ from $\mathcal{X}$ to $\mathbb{T}$. Let $\Gamma, x : T$ be the environment Δ such that $x\Delta = T$ and $y\Delta = y\Gamma$ if $y \neq x$. Two environments Γ_1 and Γ_2 are *compatible* if, for all x, $x\Gamma_1 = x\Gamma_2$.

A type assignment is a function $\tau : \mathcal{C} \cup \mathcal{F} \to \mathbb{T}$ such that $\tau_{\mathsf{true}} = \mathsf{bool}^{\mathrm{t}}$, $\tau_{\mathsf{false}} = \mathsf{bool}^{\mathrm{f}}$ and, for all $\mathsf{s} \in \mathcal{C} \cup \mathcal{F}$, τ_{s} is closed. To every type assignment τ, we associate a typing relation $\vdash_\tau$ defined in Figure 1. Note that, in contrast with [22], the typing of u and v in (if) does not depend on t. This is because we consider strong normalization instead of weak normalization. This does not reduce the expressive power of the system since we consider conditional rewriting.

A term t is typable wrt τ if there are C, Γ, T such that $\vdash C$ and $C; \Gamma \vdash_\tau t : T$. Let $\Lambda(\tau)$ be the set of terms typable wrt τ. A term t is *simply typable* if there are Γ, T simple such that $\top; \Gamma \vdash_{\overline{\tau}} t : T$ without ($\exists$intro), ($\forall$intro), ($\exists$elim), ($\forall$elim), (sub). Let $\overline{\Lambda}(\overline{\tau})$ be the set of terms simply typable wrt $\overline{\tau}$.

Fig. 1. Typing rules

$$\text{(var)}\quad \frac{x \in \mathrm{dom}(\Gamma)}{C; \Gamma \vdash_\tau x : x\Gamma} \qquad \text{(symb)}\quad \frac{\mathsf{s} \in \mathcal{C} \cup \mathcal{F}}{C; \Gamma \vdash_\tau \mathsf{s} : \tau_{\mathsf{s}}}$$

$$\text{(abs)}\quad \frac{C; \Gamma, x : T \vdash_\tau u : U \quad x \notin \Gamma}{C; \Gamma \vdash_\tau \lambda x u : T \Rightarrow U} \qquad \text{(app)}\quad \frac{C; \Gamma \vdash_\tau t : U \Rightarrow V \quad C; \Gamma \vdash_\tau u : U}{C; \Gamma \vdash_\tau tu : V}$$

$$\text{(pair)}\quad \frac{C; \Gamma \vdash_\tau u : U \quad C; \Gamma \vdash_\tau v : V}{C; \Gamma \vdash_\tau (u, v) : U \times V}$$

$$\text{(fst)}\quad \frac{C; \Gamma \vdash_\tau t : U \times V}{C; \Gamma \vdash_\tau \mathsf{fst}\, t : U} \qquad \text{(snd)}\quad \frac{C; \Gamma \vdash_\tau t : U \times V}{C; \Gamma \vdash_\tau \mathsf{snd}\, t : V}$$

$$\text{(if)}\quad \frac{C; \Gamma \vdash_\tau t : \mathsf{bool} \quad C; \Gamma \vdash_\tau u : T \quad C; \Gamma \vdash_\tau v : T \quad T \text{ } \exists\text{-basic}}{C; \Gamma \vdash_\tau \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v : T}$$

$$\text{(let)}\quad \frac{C; \Gamma \vdash_\tau t : T \quad C; \Gamma, x : T \vdash_\tau u : U \quad x \notin \Gamma}{C; \Gamma \vdash_\tau \mathsf{let}\, x = t\, \mathsf{in}\, u : U}$$

$$\text{($\forall$intro)}\quad \frac{C \wedge P; \Gamma \vdash_\tau t : T \quad \vdash C \supset \exists \boldsymbol{\alpha} P \quad \boldsymbol{\alpha} \notin C, \Gamma}{C; \Gamma \vdash_\tau t : \forall \boldsymbol{\alpha} P T}$$

$$\text{($\forall$elim)}\quad \frac{C; \Gamma \vdash_\tau t : \forall \boldsymbol{\alpha} P T \quad \vdash C \supset P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}}{C; \Gamma \vdash_\tau t : T^{\boldsymbol{a}}_{\boldsymbol{\alpha}}}$$

$$\text{($\exists$intro)}\quad \frac{C; \Gamma \vdash_\tau t : T^{\boldsymbol{a}}_{\boldsymbol{\alpha}} \quad \vdash C \supset P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}}{C; \Gamma \vdash_\tau t : \exists \boldsymbol{\alpha} P T}$$

$$\text{($\exists$elim)}\quad \frac{C; \Gamma \vdash_\tau t : \exists \boldsymbol{\alpha} P T \quad C \wedge P; \Gamma, x : T \vdash_\tau u : U \quad \vdash C \supset \exists \boldsymbol{\alpha} P \quad \boldsymbol{\alpha}, x \notin C, \Gamma, U}{C; \Gamma \vdash_\tau \mathsf{let}\, x = t\, \mathsf{in}\, u : U}$$

$$\text{(sub)}\quad \frac{C; \Gamma \vdash_\tau t : T \quad C \vdash T \leq T'}{C; \Gamma \vdash_\tau t : T'}$$

Example 1. Consider the symbols $\mathsf{append} : \forall\beta\gamma\mathsf{L}^\beta \Rightarrow \mathsf{L}^\gamma \Rightarrow \mathsf{L}^{\beta+\gamma}$ and $\mathsf{pivot} : \mathsf{N} \Rightarrow \forall\alpha\mathsf{L}^\alpha \Rightarrow \exists\beta\gamma(\alpha = \beta + \gamma)\mathsf{L}^\beta \times \mathsf{L}^\gamma$. Let $\Gamma = x : \mathsf{N}$, $l : \mathsf{L}^\alpha$, $u = (\mathsf{let}\, z = t\, \mathsf{in}\, v)$, $t = \mathsf{pivot}\, x\, l$ and $v = \mathsf{append}\,(\mathsf{fst}\, z)(\mathsf{snd}\, z)$. Then, $\top; \Gamma \vdash t : \exists\beta\gamma(\alpha = \beta+\gamma)\mathsf{L}^\beta \times \mathsf{L}^\gamma$ and $\alpha = \beta + \gamma$; $\Gamma, z : \mathsf{L}^\beta \times \mathsf{L}^\gamma \vdash v : \mathsf{L}^\alpha$. Thus, by ($\exists$elim), $\Gamma \vdash u : \mathsf{L}^\alpha$.

Rewriting. Let $\to_\beta$ be the smallest relation stable by context containing the *head-β-reduction relation* $\to_{\beta h}$ defined as follows:

$$\begin{array}{rlrlrl} (\lambda x u)t & \to_{\beta h} u_x^t & \mathsf{fst}\,(u,v) & \to_{\beta h} u & \mathsf{if\,true\,then}\, u\, \mathsf{else}\, v & \to_{\beta h} u \\ \mathsf{let}\, x = t\, \mathsf{in}\, u & \to_{\beta h} u_x^t & \mathsf{snd}\,(u,v) & \to_{\beta h} v & \mathsf{if\,false\,then}\, u\, \mathsf{else}\, v & \to_{\beta h} v \end{array}$$

A *conditional rewrite rule* is an expression of the form $\boldsymbol{t} = \mathbf{c} \supset l \to r$ such that l is of the form $\mathsf{f}\boldsymbol{l}$, $\boldsymbol{l}$ are patterns, $\mathbf{c} \in \{\mathsf{true}, \mathsf{false}\}$ and $\mathrm{FV}(r, \boldsymbol{t}) \subseteq \mathrm{FV}(l)$. A rule $\boldsymbol{t} = \mathbf{c} \supset l \to r$ *defines* $\mathsf{f} \in \mathcal{F}$ if l is of the form $\mathsf{f}\boldsymbol{l}$. In the following, we assume given a set $\mathcal{R}$ of rules. The associated rewrite relation is the smallest relation $\to_\mathcal{R}$ stable by context and substitution such that, for all $\boldsymbol{t} = \mathbf{c} \supset l \to r \in \mathcal{R}$, $l\sigma \to_\mathcal{R} r\sigma$ whenever $\boldsymbol{t}\sigma \to^* \mathbf{c}$, where $\to^*$ is the reflexive and transitive closure of $\to = \to_\beta \cup \to_\mathcal{R}$.

Our goal is to prove the strong normalization of $\to = \to_\beta \cup \to_\mathcal{R}$ on the set of simply typable terms $\overline{\Lambda}(\overline{\tau})$.

Assumption: We assume that $\to$ is locally confluent.

Hence, any strongly normalizing term t has a unique normal form $t{\downarrow}$. Note that $\to$ is locally confluent whenever $\to_\mathcal{R}$ so is. See [10] for general conditions on the confluence of β-reduction and higher-order conditional rewriting.

It should be noted that ($\exists$elim) makes subject reduction fail. For instance, with $\Gamma = x : \exists\alpha\mathsf{N}^\alpha, y : \forall\alpha\mathsf{N}^\alpha \Rightarrow \exists\beta\mathsf{N}^\beta$, we have $\top; \Gamma \vdash \mathsf{let}\, z = x\, \mathsf{in}\, yz : \exists\beta\mathsf{N}^\beta$ while yx is not typable in $\top; \Gamma$. It could be fixed by replacing in ($\exists$elim) $\mathsf{let}\, x = t\, \mathsf{in}\, u$ by u_x^t. It does not matter since our termination proof technique does not need subject reduction. Note however that subject reduction holds on simply typed terms.

An example of higher-order conditional rule is given by the following definition of $\mathsf{filter} : (\mathsf{N} \Rightarrow \mathsf{N}) \Rightarrow \forall\alpha\mathsf{L}^\alpha \Rightarrow \exists\beta(\beta \le \alpha)\mathsf{L}^\beta$:

$$\begin{array}{rcl} \mathsf{filter}\, f\, \mathsf{nil} & \to & \mathsf{nil} \\ f\, x = \mathsf{true} \supset \mathsf{filter}\, f (\mathsf{cons}\, x\, l) & \to & \mathsf{cons}\,(f\, x)\,(\mathsf{filter}\, f\, l) \\ f\, x = \mathsf{false} \supset \mathsf{filter}\, f (\mathsf{cons}\, x\, l) & \to & \mathsf{filter}\, f\, l \end{array}$$

3 Type-checking algorithm

Type-checking is the following problem: given τ, C, Γ, t and T, do we have C satisfiable and $C; \Gamma \vdash_\tau t : T$?

Because of the rules ($\exists$elim) and (conv), type-checking does not seem to be decidable. Similarly, in [22], the elaboration process is not complete. It is however possible to give an algorithm that either succeed or fails, a failure meaning that

we don't know. To this end, we inductively define in Figure 2 two relations in the style of bi-directional type inference [12, 2]. In the type inference relation $C; \Gamma \vdash t \uparrow T$, C and T are produced according to Γ and t. In the type checking relation $C; \Gamma \vdash t \downarrow T$, C is produced according to Γ, t and T. An actual algorithm is a strategy for applying the rules defining these relations.

Let $\overline{\mathbb{C}}$ be the closure of $\mathbb{C}$ by conjunction, implication, existential and universal quantification. If one starts with $C \in \mathbb{C}$, then the constraints generated by such an algorithm are in $\mathbb{C}$ too. Hence, if $\mathbb{C}$ only contains linear inequalities, then $\overline{\mathbb{C}}$ are formulas of Presburger arithmetic which is known to be decidable [18] and whose complexity is doubly exponential in the size of the formula [13]. This high complexity is not so important in our case since the terms we intend to consider are small (rule right-hand sides). It would be however interesting to study in more details the complexity of type-checking wrt $\mathbb{C}$.

For proving the correctness of the rule ($\downarrow\exists$intro), we need to assume that the size expression language $\mathcal{A}$ is complete wrt the interpretation domains $\mathcal{D}_s$, that is, to every $\mathfrak{a} \in \mathcal{D}_s$ corresponds a closed term $a \in \mathcal{A}$ whose denotation in $\mathcal{D}_s$ is $\mathfrak{a}$. Note that this is indeed the case when $\mathcal{D}_s = \mathbb{N}$ and $\mathcal{A}$ contains 0, 1 and $+$.

See Example 3 at the end of the paper for an example of derivation.

Theorem 1. *Consider the rules of Figure 2. If $C; \Gamma \vdash^? t : T$, then C is satisfiable and $C; \Gamma \vdash t : T$.*

Proof. First, one can easily check that, for every rule, if the constraint in the conclusion is satisfiable, then the constraints in the premises are satisfiable too. Then, we prove that, if C is satisfiable and $C; \Gamma \vdash t \uparrow T$ or $C; \Gamma \vdash t \downarrow T$, then $C; \Gamma \vdash t : T$. We only detail some cases.

- ($\uparrow\exists$elim) Let $E = C \wedge \exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset D)$. Since $E \supset C$ and $(E \wedge P) \supset D$, by induction hypothesis and weakening, $E; \Gamma \vdash t : \exists \boldsymbol{\alpha} P T$ and $E \wedge P; \Gamma \vdash u : U$. Since $(E \wedge P) \supset P$, by ($\exists$intro), $E \wedge P; \Gamma \vdash u : \exists \boldsymbol{\alpha} P U$. Since $E \supset \exists \boldsymbol{\alpha} P$ and $\boldsymbol{\alpha} \notin \exists \boldsymbol{\alpha} P U$, by ($\exists$elim), $E; \Gamma \vdash \mathsf{let}\, x = t \,\mathsf{in}\, u : \exists \boldsymbol{\alpha} P U$.
- ($\downarrow\forall$intro) Let $E = \exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset C)$. Since $(E \wedge P) \supset C$, by induction hypothesis and weakening, $E \wedge P; \Gamma \vdash t : T$. Since $E \supset \exists \boldsymbol{\alpha} P$, we can conclude by ($\forall$intro).
- ($\downarrow\forall$elim) Let $E = C \wedge P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}$. By induction hypothesis and weakening, $E; \Gamma \vdash t : \forall \boldsymbol{\alpha} P T$. Since $E \supset P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}$, we can conclude by ($\forall$elim).
- ($\downarrow\exists$intro) Let $E = \exists \boldsymbol{\alpha}(C \wedge P)$. Since E is satisfiable, C is satisfiable too. By completeness, there is a such that $F = C^{\boldsymbol{a}}_{\boldsymbol{\alpha}} \wedge P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}$ is satisfiable. By induction hypothesis, $C; \Gamma \vdash t : T$. By substitution and weakening, $F; \Gamma \vdash t : T^{\boldsymbol{a}}_{\boldsymbol{\alpha}}$. Since $F \supset P^{\boldsymbol{a}}_{\boldsymbol{\alpha}}$, by ($\exists$intro), $F; \Gamma \vdash t : \exists \boldsymbol{\alpha} P T$. Since $E \supset F$, we can conclude by weakening.
- ($\downarrow\exists$elim) Let $E = C \wedge \exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset D)$. Since $E \supset C$ and $(E \wedge P) \supset D$, by induction hypothesis and weakening, $E; \Gamma \vdash t : \exists \boldsymbol{\alpha} P T$ and $E \wedge P; \Gamma \vdash u : U$. Since $E \supset \exists \boldsymbol{\alpha} P$ and $\boldsymbol{\alpha} \notin U$, by ($\exists$elim), $E; \Gamma \vdash \mathsf{let}\, x = t \,\mathsf{in}\, u : U$. □

Fig. 2. Rules for deciding type-checking

$$\text{(type-check)}\quad \frac{D;\Gamma \vdash t \downarrow T \quad \vdash C \supset D \quad C \text{ satisfiable}}{C;\Gamma \vdash^{?} t : T}$$

$$\text{(}\uparrow\text{var)}\quad \frac{x \in \mathrm{dom}(\Gamma)}{\top;\Gamma \vdash x \uparrow x\Gamma} \qquad \text{(}\uparrow\text{symb)}\quad \top;\Gamma \vdash \mathsf{s} \uparrow \tau_{\mathsf{s}}$$

$$\text{(}\uparrow\text{app)}\quad \frac{C;\Gamma \vdash t \uparrow U \Rightarrow V \quad D;\Gamma \vdash u \downarrow U}{C \wedge D;\Gamma \vdash tu \uparrow V}$$

$$\text{(}\uparrow\text{pair)}\quad \frac{C;\Gamma \vdash u \uparrow U \quad D;\Gamma \vdash v \uparrow V}{C \wedge D;\Gamma \vdash (u,v) \uparrow U \times V}$$

$$\text{(}\uparrow\text{fst)}\quad \frac{C;\Gamma \vdash t \uparrow U \times V}{C;\Gamma \vdash \mathsf{fst}\, t \uparrow U} \qquad \text{(}\uparrow\text{snd)}\quad \frac{C;\Gamma \vdash t \uparrow U \times V}{C;\Gamma \vdash \mathsf{snd}\, t \uparrow V}$$

$$\text{(}\uparrow\text{let)}\quad \frac{C;\Gamma \vdash t \uparrow T \quad D;\Gamma, x : T \vdash u \uparrow U}{C \wedge D;\Gamma \vdash \mathsf{let}\, x = t\, \mathsf{in}\, u \uparrow U}$$

$$\text{(}\uparrow\forall\text{elim)}\quad \frac{C;\Gamma \vdash t \uparrow \forall \boldsymbol{\alpha} PT \quad \boldsymbol{\alpha} \notin C,\Gamma}{C \wedge P;\Gamma \vdash t \uparrow T}$$

$$\text{(}\uparrow\exists\text{elim)}\quad \frac{C;\Gamma \vdash t \uparrow \exists \boldsymbol{\alpha} PT \quad D;\Gamma, x : T \vdash u \uparrow U \quad x \notin \Gamma \quad \boldsymbol{\alpha} \notin C,\Gamma}{C \wedge \exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset D);\Gamma \vdash \mathsf{let}\, x = t\, \mathsf{in}\, u \uparrow \exists \boldsymbol{\alpha} PU}$$

$$\text{(}\downarrow\text{abs)}\quad \frac{C;\Gamma, x : T \vdash u \downarrow U \quad x \notin \Gamma}{C;\Gamma \vdash \lambda x u \downarrow T \Rightarrow U}$$

$$\text{(}\downarrow\text{if)}\quad \frac{C;\Gamma \vdash t \downarrow \exists \alpha \mathsf{bool}^{\alpha} \quad D;\Gamma \vdash u \downarrow T \quad E;\Gamma \vdash v \downarrow T \quad T\ \exists\text{-basic}}{C \wedge D \wedge E;\Gamma \vdash \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \downarrow T}$$

$$\text{(}\downarrow\forall\text{intro)}\quad \frac{C;\Gamma \vdash t \downarrow T \quad \boldsymbol{\alpha} \notin \Gamma}{\exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset C);\Gamma \vdash t \downarrow \forall \boldsymbol{\alpha} PT}$$

$$\text{(}\downarrow\forall\text{elim)}\quad \frac{C;\Gamma \vdash t \uparrow \forall \boldsymbol{\alpha} PT}{C \wedge P^{\boldsymbol{\alpha}}_{\boldsymbol{\alpha}};\Gamma \vdash t \downarrow T^{\boldsymbol{\alpha}}_{\boldsymbol{\alpha}}}$$

$$\text{(}\downarrow\exists\text{intro)}\quad \frac{C;\Gamma \vdash t \downarrow T \quad \boldsymbol{\alpha} \notin \Gamma}{\exists \boldsymbol{\alpha}(C \wedge P);\Gamma \vdash t \downarrow \exists \boldsymbol{\alpha} PT}$$

$$\text{(}\downarrow\exists\text{elim)}\quad \frac{C;\Gamma \vdash t \uparrow \exists \boldsymbol{\alpha} PT \quad D;\Gamma, x : T \vdash u \downarrow U \quad \boldsymbol{\alpha} \notin C,\Gamma,U}{C \wedge \exists \boldsymbol{\alpha} P \wedge \forall \boldsymbol{\alpha}(P \supset D);\Gamma \vdash \mathsf{let}\, x = t\, \mathsf{in}\, u \downarrow U}$$

$$\text{(}\downarrow\text{sub)}\quad \frac{C;\Gamma \vdash t \uparrow T'}{C \wedge (\!|T' \leq T|\!);\Gamma \vdash t \downarrow T}$$

4 Termination proof technique

In this section, we present a general method for proving the strong normalization of β-reduction and rewriting on well-typed terms. It is based on Tait's method for proving the strong normalization of β-reduction [19]. The idea is to interpret types by particular sets of strongly normalizing terms, called saturated, and prove that every well-typed term belongs to the interpretation of its type.

Following [2], we define the *weak-head-β-reduction relation* $\to_{\beta wh}$ as the relation such that $E[t] \to_{\beta wh} E[u]$ iff $t \to_{\beta h} u$ and $E \in \mathcal{E}$, where the set of *elimination contexts* $\mathcal{E}$ is inductively defined as follows:

$$E \in \mathcal{E} ::= [] \mid E\,t \mid \mathsf{fst}\ E \mid \mathsf{snd}\ E$$

Definition 1 (Saturated sets). *The set* SAT *of* saturated sets *is the set of all the sets of terms S such that:*
(1) If $t \in S$, then $t \in \mathrm{SN}$.
(2) If $t \in S$ and $t \to t'$, then $t' \in S$.
(3) If $E[x] \in \mathrm{SN}$, then $E[x] \in S$.
(4) If $t \in \mathrm{SN}$, $t \to_{\beta h} t'$ and $E[t'] \in S$, then $E[t] \in S$.
We also define the following operations on sets of terms:
$- \ S_1 \Rightarrow S_2 = \{t \in \mathcal{T} \mid \forall u \in S_1,\ tu \in S_2\}$
$- \ S_1 \times S_2 = \{t \in \mathcal{T} \mid \mathsf{fst}\,t \in S_1 \wedge \mathsf{snd}\,t \in S_2\}$
Let $\mathcal{N}$ be the set of terms of the form $\mathsf{f}\boldsymbol{t}$, $\mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v$, $\mathsf{fst}\,t$ or $\mathsf{snd}\,t$. A saturated set S has the neutral term property *if $s \in S$ whenever $s \in \mathcal{N}$ and $\to(s) \subseteq S$.*

Lemma 1. SAT *is a complete lattice for inclusion with $\bigcup$ as lub, $\bigcap$ as glb and* SN *as greatest element. It is also stable by $\Rightarrow$ and $\times$.*

All this is more or less well known. See for instance [2]. The key difference with the first author work [8] is that we use saturated sets instead of reducibility candidates. See [14] for a comparison between the two kinds of sets. With reducibility candidates, (4) is replaced by the neutral term property.

Reducibility candidates are saturated but the converse does not hold since candidates are not stable by union. Hence, with candidates, $\exists \boldsymbol{\alpha} PT$ cannot be interpreted as an union, which is essential if one wants to interpret $\mathsf{B}^{\mathfrak{a}}$ as the set of terms of size $\mathfrak{a}$ in order to give precise types to function symbols.

However, reducibility candidates extend well to rewriting and polymorphism since, for proving that $\mathsf{f}\boldsymbol{t} \in S$, it suffices to prove that $\to(\mathsf{f}\boldsymbol{t}) \subseteq S$. In Lemma 2, we prove that this property still holds with saturated sets when S is the interpretation of an existentially quantified basic type.

Definition 2 (Interpretation of types). *A base type interpretation is a function I which, to every pair $(\mathsf{B}, \mathfrak{a})$ with $\mathsf{B} \neq \mathsf{bool}$, associates a set $I^{\mathfrak{a}}_{\mathsf{B}} \in$ SAT. We extend I to* bool *by taking $I^{\mathfrak{a}}_{\mathsf{bool}} = \{t \in \mathrm{SN} \mid t{\downarrow} \neq \mathfrak{a}^*\}$. Given such an interpretation, types are interpreted by saturated sets as follows:*
$- \ [\![\mathsf{B}^a]\!]^I_\mu = I^{a\mu}_{\mathsf{B}}$

– $[\![U \times V]\!]^I_\mu = [\![U]\!]^I_\mu \times [\![V]\!]^I_\mu$
– $[\![U \Rightarrow V]\!]^I_\mu = [\![U]\!]^I_\mu \Rightarrow [\![V]\!]^I_\mu$
– $[\![\forall \boldsymbol{\alpha} PT]\!]^I_\mu = \bigcap_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}} \models P} [\![T]\!]^I_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}}}$ *if* $\vdash \exists \boldsymbol{\alpha} P$, $[\![\forall \boldsymbol{\alpha} PT]\!]^I_\mu = \mathrm{SN}$ *otherwise*
– $[\![\exists \boldsymbol{\alpha} PT]\!]^I_\mu = \bigcup_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}} \models P} [\![T]\!]^I_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}}}$ *if* $\vdash \exists \boldsymbol{\alpha} P$, $[\![\exists \boldsymbol{\alpha} PT]\!]^I_\mu = \bigcap \mathrm{SAT}$ *otherwise*

Let $I^\omega_{\mathsf{B}} = [\![\exists \alpha \mathsf{B}^\alpha]\!]$. *A symbol* $\mathsf{s} \in \mathcal{C} \cup \mathcal{F}$ *is* computable *if* $\mathsf{s} \in [\![\tau_\mathsf{s}]\!]^I$. *A pair* (μ, σ) *is* valid *for* $C; \Gamma$, *written* $(\mu, \sigma) \models C; \Gamma$, *if* $\mu \models C$ *and, for all* $x \in \mathrm{dom}(\Gamma)$, $x\sigma \in [\![x\Gamma]\!]^I_\mu$. *A base type interpretation* I *is valid if every constructor is computable and, for every* $\exists$*-basic type* T, $[\![T]\!]^I_\mu$ *has the neutral term property.*

Note that $I^{\mathfrak{a}}_{\mathsf{bool}} \in \mathrm{SAT}$ has the neutral term property and $[\![T\varphi]\!]^I_\mu = [\![T]\!]^I_{\varphi\mu}$.

Theorem 2. *Assume that* I *is a valid base type interpretation and every* $\mathsf{f} \in \mathcal{F}$ *is computable. If* $C; \Gamma \vdash t : T$ *and* $(\mu, \sigma) \models C; \Gamma$, *then* $t\sigma \in [\![T]\!]^I_\mu$.

Proof. By induction on $C; \Gamma \vdash t : T$. We only detail some cases.

(abs) We must prove that $s = (\lambda x u)\sigma \in [\![T \Rightarrow U]\!]^I_\mu$. Wlog, we can assume that $x \notin \sigma$. Then, $s = \lambda x (u\sigma)$. Let $t \in [\![T]\!]^I_\mu$. We must prove that $st \in [\![U]\!]^I_\mu$. By induction hypothesis, $u\sigma \in [\![U]\!]^I_\mu$. Let now $\sigma' = \sigma +^t_x$. Since $(\mu, \sigma') \models C; \Gamma, x : T$, by induction hypothesis, $u\sigma' \in [\![U]\!]^I_\mu$. Hence, $st \in \mathrm{SN}$ since, by induction on $(u\sigma, t)$ with $\to_{\mathrm{lex}}$ as well-founded ordering, $\to(st) \subseteq \mathrm{SN}$. Therefore, $st \in [\![U]\!]^I_\mu$ since $st \to_{\beta h} u\sigma' \in [\![U]\!]^I_\mu$ and $st \in \mathrm{SN}$.

(if) Let $s = (\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v)\sigma$. By induction hypothesis, $t\sigma \in I^\omega_{\mathsf{bool}}$ and $t_i\sigma \in [\![T]\!]^I_\mu$. Since $s \in \mathcal{N}$ and T is an $\exists$-basic type, by the neutral term property, it suffices to prove that $\to(s) \subseteq [\![T]\!]^I_\mu$. This follows by induction on $(t\sigma, u\sigma, v\sigma)$ with $\to_{\mathrm{lex}}$ as well-founded ordering.

($\exists$elim) We must prove that $s = (\mathsf{let}\, x = t\, \mathsf{in}\, u)\sigma \in [\![U]\!]^I_\mu$. Wlog, we can assume that $x \notin \sigma$. Then, $s = \mathsf{let}\, x = t\sigma\, \mathsf{in}\, u\sigma$. Let $\sigma' = \sigma^{t\sigma}_x$. By induction hypothesis, $t\sigma \in [\![\exists \boldsymbol{\alpha} PT]\!]^I_\mu$. Since $\vdash C \supset \exists \boldsymbol{\alpha} P$, there is $\mathfrak{a}$ such that $\mu +^{\mathfrak{a}}_{\boldsymbol{\alpha}} \models P$ and $t\sigma \in [\![T]\!]^I_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}}}$. Therefore, by induction hypothesis, $u\sigma' \in [\![U]\!]^I_{\mu + \frac{\mathfrak{a}}{\boldsymbol{\alpha}}} = [\![U]\!]^I_\mu$.

(sub) By induction on T and T', one can easily prove that $[\![T]\!]^I_\mu \subseteq [\![U]\!]^I_\mu$ whenever $\mu \models (\!| T \leq U |\!)$. □

Corollary 1. *Assume that* I *is a valid base type interpretation and every* $\mathsf{f} \in \mathcal{F}$ *is computable. Then,* $\to$ *is strongly normalizing on* $\Lambda(\tau)$.

Corollary 2. *Assume that, for all* $\mathsf{s} \in \mathcal{C} \cup \mathcal{F}$, τ_s *is of the form* $\boldsymbol{T} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow T$ *with* $\boldsymbol{T}$ *simple,* B *basic and* T *an* $\exists$*-basic type. If every symbol is computable, then* $\to$ *is strongly normalizing on* $\overline{\Lambda}(\overline{\tau})$.

Proof. It suffices to prove that, for all s, $\mathsf{s} \in [\![\overline{\tau_\mathsf{s}}]\!]^I$. We have $\overline{\tau_\mathsf{s}} = \boldsymbol{T} \Rightarrow \mathbf{B} \Rightarrow \overline{B}$. Let $\boldsymbol{t} \in [\![\boldsymbol{T}]\!]^I$ and $\boldsymbol{u} \in I^\omega_{\mathbf{B}}$. We must prove that $\mathsf{f}\boldsymbol{tu} \in [\![\overline{B}]\!]^I$. There is $\boldsymbol{\alpha}\mu$ such that $\boldsymbol{u} \in I^{\boldsymbol{\alpha}\mu}_{\mathbf{B}}$. Assume that $T = \forall \boldsymbol{\delta} \boldsymbol{P} B$. Since $\mathsf{f} : \boldsymbol{T} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow T$ is computable, $\mathsf{f}\boldsymbol{tu} \in [\![T]\!]^I_\mu = \bigcup_{\mu + \frac{\mathfrak{d}}{\boldsymbol{\delta}} \models \boldsymbol{P}} [\![B]\!]^I_{\mu + \frac{\mathfrak{d}}{\boldsymbol{\delta}}}$. Let $\nu = \mu +^{\mathfrak{d}}_{\boldsymbol{\delta}} \models \boldsymbol{P}$. We are left to prove that $[\![B]\!]^I_\nu \subseteq [\![\overline{B}]\!]^I$. We proceed by induction on B. □

5 Termination criterion

We now provide conditions to obtain the computability of defined symbols.

A *precedence* is a quasi-ordering $\geq$ whose strict part $> = \geq \setminus \leq$ is well-founded. Let $\simeq = \geq \cap \leq$ be its associated equivalence relation. We assume given a precedence $\geq_{\mathcal{B}}$ on $\mathcal{B}$ and a precedence $\geq_{\mathcal{F}}$ on $\mathcal{F}$. We are going to define some base type interpretation and prove that every function symbol is computable by induction on these precedences.

Assumption: For all $\mathsf{c} \in \mathcal{C}$, we assume that τ_{c} is of the form[1] $\mathbf{C} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow \mathsf{B}^a$ with $\mathbf{C} <_{\mathcal{B}} \mathsf{B}$, $\mathbf{B} \simeq_{\mathcal{B}} \mathsf{B}$, $a = 0$ if $|\boldsymbol{\alpha}| = 0$, and $a = 1 + max(\boldsymbol{\alpha})$ if $|\boldsymbol{\alpha}| > 0$.

Example 2. The type N of natural numbers has constructors $\mathsf{0} : \mathsf{N}^0$ and $\mathsf{s} : \forall \alpha \mathsf{N}^\alpha \Rightarrow \mathsf{N}^{\alpha+1}$. The type L of lists has constructors $\mathsf{nil} : \mathsf{L}^0$ and $\mathsf{cons} : \mathsf{N} \Rightarrow \forall \alpha \mathsf{L}^\alpha \Rightarrow \mathsf{L}^{\alpha+1}$. The type T of binary trees has constructors $\mathsf{leaf} : \mathsf{N} \Rightarrow \mathsf{T}^0$ and $\mathsf{node} : \forall \alpha\beta \mathsf{T}^\alpha \Rightarrow \mathsf{T}^\beta \Rightarrow \mathsf{T}^{1+max(\alpha,\beta)}$.

We define the base type interpretation as follows:

- $I^0_{\mathsf{B}} = \{t \in \mathrm{SN} \mid \forall \mathsf{c} : \mathbf{C} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow \mathsf{B}^a, \forall \boldsymbol{tu}, |\boldsymbol{t}| = |\mathbf{C}| \wedge |\boldsymbol{u}| = |\boldsymbol{\alpha}| \wedge t \rightarrow^* \mathsf{c}\boldsymbol{tu} \Rightarrow \boldsymbol{t} \in I^\omega_{\mathbf{C}} \wedge |\boldsymbol{\alpha}| = a = 0\}$
- $I^{\mathfrak{a}+1}_{\mathsf{B}} = \{t \in \mathrm{SN} \mid \forall \mathsf{c} : \mathbf{C} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow \mathsf{B}^a, \forall \boldsymbol{tu}, |\boldsymbol{t}| = |\mathbf{C}| \wedge |\boldsymbol{u}| = |\boldsymbol{\alpha}| \wedge t \rightarrow^* \mathsf{c}\boldsymbol{tu} \Rightarrow \boldsymbol{t} \in I^\omega_{\mathbf{C}} \wedge a = 1 + max(\boldsymbol{\alpha}) \wedge (\exists \mathfrak{b})\, \mathfrak{a} = max(\mathfrak{b}) \wedge \boldsymbol{u} \in I^{\mathfrak{b}}_{\mathbf{B}}\}$

Lemma 2. *I is a valid base type interpretation.*

Proof. One can easily check that $I^{\mathfrak{a}}_{\mathsf{b}}$ is saturated and that every constructor is computable. We now prove that $[\![T]\!]^I_\mu$ has the neutral term property whenever T is $\exists$-basic.

We first remark that, if $t \in \mathrm{SN}$ and $t \rightarrow^* t' \in I^{\mathfrak{a}}_{\mathsf{B}}$, then $t \in I^{\mathfrak{a}}_{\mathsf{B}}$. We prove it by induction on $(\mathsf{B}, \mathfrak{a})$ with $(>_{\mathcal{B}}, >_{\mathcal{D}_{\kappa_{\mathcal{B}}}})_{\mathrm{lex}}$ as well-founded ordering. Let $\mathsf{c} : \mathbf{C} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow \mathsf{B}^a$, $\boldsymbol{t}$ and $\boldsymbol{u}$ such that $|\boldsymbol{t}| = |\mathbf{C}|$, $|\boldsymbol{u}| = |\boldsymbol{\alpha}|$ and $t \rightarrow^* \mathsf{c}\boldsymbol{tu}$. By confluence, $t' \rightarrow^* \mathsf{c}\boldsymbol{t'u'}$ with $\boldsymbol{tu} \rightarrow^* \boldsymbol{t'u'}$. We proceed by case on $\mathfrak{a}$.

- $\mathfrak{a} = \mathfrak{t}$. Then, $t' \not\rightarrow^* \mathsf{false}$. Hence, $t \not\rightarrow^* \mathsf{false}$ and $t \in I^{\mathfrak{a}}_{\mathsf{B}}$.
- $\mathfrak{a} = \mathfrak{f}$. Idem.
- $\mathfrak{a} = 0$. Since $t' \in I^{\mathfrak{a}}_{\mathsf{B}}$, $\boldsymbol{t'} \in I^\omega_{\mathbf{C}}$ and $|\boldsymbol{\alpha}| = a = 0$. Since $\mathbf{C} <_{\mathcal{B}} \mathsf{B}$, by induction hypothesis, $\boldsymbol{t} \in I^\omega_{\mathbf{C}}$. Thus, $t \in I^{\mathfrak{a}}_{\mathsf{B}}$.
- $\mathfrak{a} > 0$. Since $t' \in I^{\mathfrak{a}}_{\mathsf{B}}$, $\boldsymbol{t'} \in I^\omega_{\mathbf{C}}$, $a = 1 + max(\boldsymbol{\alpha})$ and there are $\mathfrak{b}$ such that $\mathfrak{a} = 1 + max(\mathfrak{b})$ and $\boldsymbol{u'} \in I^{\mathfrak{b}}_{\mathbf{B}}$. Since $\mathbf{C} <_{\mathcal{B}} \mathsf{B}$ and $\mathfrak{b} < \mathfrak{a}$, by induction hypothesis, $\boldsymbol{t} \in I^\omega_{\mathbf{C}}$ and $\boldsymbol{u} \in I^{\mathfrak{b}}_{\mathbf{B}}$. Thus, $t \in I^{\mathfrak{a}}_{\mathsf{B}}$.

Let now $T = \exists \boldsymbol{\alpha} \boldsymbol{P} B$ be an $\exists$-basic type. We have $S = \bigcup_{\mu +^{\mathfrak{a}}_{\boldsymbol{\alpha}} \models \boldsymbol{P}} [\![B]\!]^I_{\mu +^{\mathfrak{a}}_{\boldsymbol{\alpha}}}$. We first prove that there are $\mathfrak{a}$ such that $\nu = \mu +^{\mathfrak{a}}_{\boldsymbol{\alpha}} \models \boldsymbol{P}$ and $\rightarrow(s) \subseteq S' = [\![B]\!]^I_\nu$. If $\rightarrow(s) = \emptyset$, this is immediate. So, assume that there is $t \in \rightarrow(s)$. Since $t \in S$, there are $\mathfrak{a}$ such that $\nu = \mu +^{\mathfrak{a}}_{\boldsymbol{\alpha}} \models \boldsymbol{P}$ and $t \in S' = [\![B]\!]^I_\nu$. Let now $u \in \rightarrow(s)$. By

[1] The order of types is not relevant. We take this order for the sake of simplicity.

confluence, there is v such that $t, u \to^* v$. Since $t \in S'$, we have $v \in S'$. Thus, $u \in S'$ too. Hence, $\to(s) \subseteq S'$.

We now prove that $s \in S'$ whenever $\to(s) \subseteq S'$ by induction on B. □

Fig. 3. Matching constraints

$$(1) \quad \alpha = \varepsilon_x; x : \mathsf{B}^{\varepsilon_x} \rightsquigarrow x : \mathsf{B}^{\alpha}$$

$$(2) \quad \frac{\mathsf{c} : \boldsymbol{T} \Rightarrow \mathsf{B}^0 \quad \mathsf{B} \neq \mathsf{bool}}{\alpha = 0; \boldsymbol{x} : \boldsymbol{T} \rightsquigarrow \mathsf{c}\boldsymbol{x} : \mathsf{B}^{\alpha}} \qquad (2') \quad \frac{\mathsf{c} : \mathsf{bool}^{\mathsf{c}^*}}{\alpha = \mathsf{c}^*; \emptyset \rightsquigarrow \mathsf{c} : \mathsf{bool}^{\alpha}}$$

$$(3) \quad \frac{c : \boldsymbol{T} \Rightarrow \forall \boldsymbol{\alpha}\mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow \mathsf{B}^{1+max(\boldsymbol{\alpha})} \quad \boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{\Gamma} \rightsquigarrow \boldsymbol{u} : \mathbf{B}^{\boldsymbol{\alpha}} \quad \alpha \notin \boldsymbol{\alpha} \quad \boldsymbol{x} : \boldsymbol{T}, \boldsymbol{\Gamma} \text{ are compatible}}{\alpha = 1 + max(\boldsymbol{a}); \boldsymbol{x} : \boldsymbol{T}, \boldsymbol{\Gamma} \rightsquigarrow \mathsf{c}\boldsymbol{x}\boldsymbol{u} : \mathsf{B}^{\alpha}}$$

Lemma 3. *We assume given an injection ε from term variables to size variables. Consider the rules of Figure 3. If $\alpha = a; \Gamma \rightsquigarrow t : \mathsf{B}^{\alpha}$ and $t\sigma \in I_{\mathsf{B}}^{\alpha\mu}$, then there is ν such that $(\mu + \nu, \sigma) \models \alpha = a; \Gamma$.*

Proof. We say that $\mathfrak{a}$ is minimal for $t \in [\![\mathsf{B}]\!]^{\omega}$ if $t \in [\![\mathsf{B}]\!]^{\mathfrak{a}}$ and, for all $\mathfrak{b} < \mathfrak{a}$, $t \notin [\![\mathsf{B}]\!]^{\mathfrak{b}}$. We prove the lemma by induction on $\alpha = a; \Gamma \rightsquigarrow t : \mathsf{B}^{\alpha}$ with the additional requirement that ν is minimal whenever μ so is.

(1) It suffices to take $\varepsilon_x \nu = \alpha\mu$.

(2) and (2') It suffices to take $\nu = \emptyset$.

(3) We have $t\sigma = c\boldsymbol{x}\sigma\boldsymbol{u}\sigma$. Thus, μ is minimal, $x\sigma \in [\![\boldsymbol{T}]\!]$ and there is μ' minimal such that $\boldsymbol{u}\sigma \in I_{\mathsf{B}}^{\boldsymbol{\alpha}\mu'}$ and $\alpha\mu = 1 + max(\boldsymbol{\alpha}\mu')$. Now, by induction hypothesis, there are $\boldsymbol{\nu}$ minimal such that $(\mu' + \boldsymbol{\nu}, \sigma) \models \boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{\Gamma}$. Since $\boldsymbol{\nu}$ are minimal, if $x\sigma \in I_{\mathsf{B}_i}^{\varepsilon_x \nu_i} \cap I_{\mathsf{B}_j}^{\varepsilon_x \nu_j}$, then $\varepsilon_x \nu_i = \varepsilon_x \nu_j$. Thus, we can define $\nu = \Sigma\boldsymbol{\nu}$. Since ν is minimal, we are left to prove that $(\mu + \nu, \sigma) \models \alpha = 1 + max(\boldsymbol{a}); \boldsymbol{\Gamma}$. First, we have $\mu + \nu \models \alpha = 1 + max(\boldsymbol{a})$ since $\alpha\mu = 1 + max(\boldsymbol{\alpha}\mu') = 1 + max(\boldsymbol{a}\nu)$. Second, let $x \in u_i$. Then, $x\sigma \in [\![x\Gamma]\!]^I_{\nu_i} = [\![x\Gamma]\!]^I_{x\nu}$. □

Theorem 3 (Termination criterion). *Assume that, for every $\mathsf{f} \in \mathcal{F}$:*

(1) τ_{f} is of the form $\boldsymbol{T} \Rightarrow \forall \boldsymbol{\alpha}\mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow T$ with T an $\exists$-basic type;

(2) there is a constraint $(\boldsymbol{\beta} <_{\mathsf{f}} \boldsymbol{\alpha})$ such that the ordering $\succ_{\mathsf{f}}$ defined by $\boldsymbol{\alpha}\mu \succ_{\mathsf{f}} \boldsymbol{\beta}\mu$ iff $\mu \models \boldsymbol{\beta} <_{\mathsf{f}} \boldsymbol{\alpha}$ is well-founded;

(3) for every $\mathsf{g} \simeq_{\mathcal{F}} \mathsf{f}$, τ_{g} is of the form $\boldsymbol{U} \Rightarrow \forall \boldsymbol{\alpha}\mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow U$ and $<_{\mathsf{f}} = <_{\mathsf{g}}$;

and, for every rule $\boldsymbol{t} = \mathsf{c} \supset l \to r$ defining f:

(4) l is of the form $\mathsf{f}\boldsymbol{x}\boldsymbol{l}$ with $|\boldsymbol{x}| = |\boldsymbol{T}|$ and $|\boldsymbol{l}| = |\boldsymbol{\alpha}|$;

(5) there are $\boldsymbol{\Gamma}$ compatible and $\boldsymbol{a}$ such that $\boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{\Gamma} \rightsquigarrow \boldsymbol{l} : \mathbf{B}^{\boldsymbol{\alpha}}$;

(6) every symbol occurring in r is $\leq_{\mathcal{F}} f$;

(7) $\boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{x} : \boldsymbol{T}, \boldsymbol{\Gamma} \vdash_{\tau<} \boldsymbol{t} : \mathsf{bool}^{\boldsymbol{b}}$*;*
(8) $\boldsymbol{b} = \mathbf{c}^*; \boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{x} : \boldsymbol{T}, \boldsymbol{\Gamma} \vdash_{\tau<} r : T$.
where:
(9) for every $\mathsf{g} <_{\mathcal{F}} \mathsf{f}$, $\tau_{\mathsf{g}}^{<} = \tau_{\mathsf{g}}$*;*
(10) for every $\mathsf{g} \simeq_{\mathcal{F}} \mathsf{f}$, $\tau_{\mathsf{g}}^{<} = \boldsymbol{U} \Rightarrow \forall \boldsymbol{\alpha}'(\boldsymbol{\alpha}' <_{\mathsf{f}} \boldsymbol{\alpha})\mathbf{B}^{\boldsymbol{\alpha}'} \Rightarrow U$ *with* $\boldsymbol{\alpha}' \notin \boldsymbol{\alpha}$ *whenever* $\tau_{\mathsf{g}} = \boldsymbol{U} \Rightarrow \forall \boldsymbol{\alpha}' \mathbf{B}^{\boldsymbol{\alpha}'} \Rightarrow U$.
Then, $\rightarrow$ *is strongly normalizing on* $\Lambda(\tau)$ *and* $\overline{\Lambda}(\overline{\tau})$.

Proof. We must prove that, for all $\mathsf{f} : \boldsymbol{T} \Rightarrow \forall \boldsymbol{\alpha} \mathbf{B}^{\boldsymbol{\alpha}} \Rightarrow T$, $\boldsymbol{t} \in [\![\boldsymbol{T}]\!]$, μ and $\boldsymbol{u} \in I_{\mathbf{B}}^{\boldsymbol{\alpha}\mu}$, $\mathsf{f}\boldsymbol{t}\boldsymbol{u} \in [\![T]\!]_{\mu}^{I}$. We proceed by induction on $(\mathsf{f}, \alpha\mu, \boldsymbol{t}\boldsymbol{u})$ with $(>_{\mathcal{F}}, \succ_{\mathsf{f}}, \rightarrow_{\mathrm{lex}})_{\mathrm{lex}}$ as well-founded ordering. By Lemma 2, it suffices to prove that $\rightarrow(s) \subseteq S$. If the reduction takes place in $\boldsymbol{t}\boldsymbol{u}$, we conclude by induction hypothesis. Assume now that there are $\mathsf{f}\boldsymbol{x}\boldsymbol{l} \rightarrow r \in \mathcal{R}$ and σ such that $\boldsymbol{x}\sigma = \boldsymbol{t}$ and $\boldsymbol{l}\sigma = \boldsymbol{u}$. We must prove that $r\sigma \in [\![T]\!]_{\mu}^{I}$. After Lemma 3, since $\boldsymbol{\Gamma}$ are compatible, there is ν such that $(\mu + \nu, \sigma) \models \boldsymbol{\alpha} = \boldsymbol{a}; \boldsymbol{\Gamma}$. By induction hypothesis, for all $g \leq_{\mathcal{F}} f$, $g \in [\![\tau_{\mathsf{g}}^{<}]\!]$ (considering $\boldsymbol{\alpha}$ as constants interpreted by $\boldsymbol{\alpha}\mu$). Thus, letting $\eta = \mu + \nu$, by Theorem 2, we have $\boldsymbol{t}\sigma \in I_{\mathsf{bool}}^{\boldsymbol{b}\eta}$. Since $t\sigma \rightarrow^* \mathbf{c} \in I_{\mathsf{bool}}^{\mathbf{c}^{**}}$, we have $\boldsymbol{b}\eta = \mathbf{c}^{**}$. Thus, $\eta \models \boldsymbol{b} = \mathbf{c}^*$ and, by Theorem 2 again, $r\sigma \in [\![T]\!]_{\eta}^{I} = [\![T]\!]_{\mu}^{I}$. □

The size variables $\boldsymbol{\alpha}$ in the type of f (1) represents the sizes of the recursive arguments of f. The user-defined predicate $<_{\mathsf{f}}$ in (2) expresses the measure that must decrease in recursive calls. One can for instance take lexicographic or multiset comparisons together with linear combinations of the arguments. The condition (5) provides the constraints on $\boldsymbol{\alpha}$ when a term matches the rule left hand-side $l = \mathsf{f}\boldsymbol{x}\boldsymbol{l}$. The condition (7) implies that the terms $\boldsymbol{t}$ are terminating whenever the arguments of the left hand-side so are. The condition (8) implies that the right hand-side is terminating whenever the arguments of the left hand-side so are and $\boldsymbol{t} \rightarrow^* \mathbf{c}$. The fact that $\boldsymbol{t} \rightarrow^* \mathbf{c}$ is expressed by the additional constraint $\boldsymbol{b} = \mathbf{c}^*$. Termination is ensured by doing type-checking in the system $\vdash_{\tau<}$ where, by condition (10), function symbols equivalent to f can only be applied to arguments smaller than $\boldsymbol{\alpha}$ in $<_{\mathsf{f}}$. This is in contrast with [8] where a new type system (called the computability closure) restricting the use of (app) must be introduced.

Example 3. We detail the criterion with the second rule of pivot given in the introduction. Let r be the right-hand side of the rule and u (resp. v) be the first (resp. second) branch of if in r.

We take $\mathsf{pivot} : \mathsf{N} \Rightarrow \forall \alpha \mathsf{L}^{\alpha} \Rightarrow T(\alpha)$ with $T(\alpha) = \exists \beta\gamma(\alpha = \beta + \gamma)\mathsf{L}^{\beta} \times \mathsf{L}^{\gamma}$, $<_{\mathsf{f}} = <$, $\succ_{\mathsf{f}} = >_{\mathbb{N}}$ and $\mathsf{le} : \mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{bool}$. Let $\Gamma = y : \mathsf{N}, l : \mathsf{L}^{\delta}$ and $\Delta = x : \mathsf{N}, \Gamma$.

Matching constraint: $\alpha = \delta + 1; \Gamma \rightsquigarrow \mathsf{cons}\, y\, l : \mathsf{L}^{\alpha}$ (we take $\varepsilon_l = \delta$).

We must check that $\alpha = \delta + 1; \Delta \vdash r : T(\alpha)$ with $\mathsf{pivot} : \mathsf{N} \Rightarrow \forall \alpha'(\alpha' < \alpha)\mathsf{L}^{\alpha'} \Rightarrow T(\alpha')$. Let $\Delta = \Gamma, z : \mathsf{L}^{\beta} \times \mathsf{L}^{\gamma}$.

One can easily check that $\delta < \alpha; \Gamma \vdash \mathsf{pivot}\, x\, l \uparrow T(\delta)$, $\top; \Delta \vdash \mathsf{le}\, y\, x \uparrow \mathsf{bool}$, $\top; \Delta \vdash u \uparrow \mathsf{L}^{\beta+1} \times \mathsf{L}^{\gamma}$, $\top; \Delta \vdash v \uparrow \mathsf{L}^{\beta} \times \mathsf{L}^{\gamma+1}$.

Thus, by $(\downarrow\mathrm{sub})$, $\beta + 1 = \beta' \wedge \gamma = \gamma'; \Delta \vdash u \downarrow \mathsf{L}^{\beta'} \times \mathsf{L}^{\gamma'}$ and $\beta = \beta' \wedge \gamma + 1 = \gamma'; \Delta \vdash u \downarrow \mathsf{L}^{\beta'} \times \mathsf{L}^{\gamma'}$.

By ($\downarrow\exists$intro), $D; \Delta \vdash u \downarrow T(\alpha)$ where $D = \exists\beta'\gamma'(\beta+1=\beta' \wedge \gamma=\gamma' \wedge \alpha = \beta'+\gamma')$, and $E; \Delta \vdash v \downarrow T(\alpha)$ where $E = \exists\beta'\gamma'(\beta=\beta' \wedge \gamma+1=\gamma' \wedge \alpha = \beta'+\gamma')$. Note that $D \equiv E \equiv (\alpha = \beta+\gamma+1)$.

By ($\downarrow$if), $\alpha = \beta+\gamma+1; \Delta \vdash \mathsf{if}\,(\mathsf{le}\,y\,x)\,\mathsf{then}\,u\,\mathsf{else}\,v : T(\alpha)$.

By ($\downarrow\exists$elim), $F; \Gamma \vdash r \downarrow T(\alpha)$ where $F = \delta < \alpha \wedge (\exists\beta\gamma(\alpha=\beta+\gamma)) \wedge (\forall\beta\gamma(\delta = \beta+\gamma \supset \alpha = \beta+\gamma+1))$.

Therefore, $\alpha = \delta+1; \Delta \vdash r : T(\alpha)$ if $\vdash \alpha = \delta+1 \supset F$, which is true.

Example 4. Consider the following definition of Mc Carthy's 91 function:

$$\begin{array}{lcl} \mathsf{le}\,x\,100 & = & \mathsf{true} \supset \mathsf{f}\,x \rightarrow \mathsf{f}\,(\mathsf{f}\,(\mathsf{plus}\,x\,11)) \\ \mathsf{le}\,x\,100 & = & \mathsf{false} \supset \mathsf{f}\,x \rightarrow \mathsf{minus}\,x\,10 \end{array}$$

We assume that $\mathcal{A}$ contains $le : nat \times nat \Rightarrow bool$ interpreted as expected.

We assume that $\mathsf{le} : \forall\alpha\beta\mathsf{N}^\alpha \Rightarrow \mathsf{N}^\beta \Rightarrow \mathsf{bool}^{le(\alpha,\beta)}$, $\mathsf{plus} : \forall\alpha\beta\mathsf{N}^\alpha \Rightarrow \mathsf{N}^\beta \Rightarrow \mathsf{N}^{\alpha+\beta}$, $\mathsf{minus} : \forall\alpha\beta\mathsf{N}^\alpha \Rightarrow \mathsf{N}^\beta \Rightarrow \exists\gamma P\mathsf{N}^\gamma$ with $P = (\alpha \leq \beta \wedge \gamma = 0) \vee (\alpha > \beta \wedge \alpha = \beta+\gamma)$, and $\mathsf{f} : \forall\alpha\mathsf{N}^\alpha \Rightarrow \exists\beta Q\mathsf{N}^\beta$ with $Q = (\alpha \leq 100 \wedge \beta = 91) \vee (\alpha > 100 \wedge \alpha = \beta+10)$. Taking $\Gamma = x : \mathsf{N}^\alpha$, we get that $\top; \Gamma \vdash \mathsf{le}\,x\,100 : \mathsf{bool}^{le(\alpha,100)}$. The condition $le(\alpha, 100) = \mathrm{t}$ is equivalent to $\alpha \leq 100$, hence the termination.

6 Conclusion and future work

We extended the simply typed part of [8] with conditional rewriting and explicit quantifications and constraints over size annotations. This allows to precisely describe the relation between the size of the output of a function and the size of its inputs. This also provides a powerful termination criterion for the combination of β-reduction and higher-order conditional rewriting, based on type-checking and constraint solving. To our knowledge, this is the first termination criterion for higher-order conditional rewriting taking into account conditions in termination. We plan to extend this work in various directions:

- As in [22], we did not consider constructors with recursive arguments of higher-order type since this is already studied in [8]. The integration of both works should not create too much difficulties. We already have preliminary results in this direction.
- The complexity of Presburger arithmetic is high. Although it is not so important in our case since the constraints we consider are small (rule right-hand sides are generally not very big terms), it would be interesting to study the complexity in more details, depending on the allowed size annotations.
- Our long term goal is to extend the present work to polymorphic and dependent type systems that serve as basis for proof assistants like Coq, *e.g.* the Calculus of Algebraic Constructions [9].
- We assume that constrained types of function symbols are given and check that they imply termination. It would be very interesting to infer these constraints automatically.

References

1. A. Abel. Termination and productivity checking with continuous types. In *Proc. of TLCA'03*, LNCS 2701.
2. A. Abel. Termination checking with types. *Theoretical Informatics and Applications*, 38(4):277–319, 2004.
3. G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004.
4. G. Barthe, B. Grégoire, and F. Pastawski. Practical inference for type-based termination in a polymorphic setting. In *Proc. of TLCA'05*, LNCS 3461.
5. F. Blanqui. Decidability of type-checking in the Calculus of Algebraic Constructions with size annotations. In *Proc. of CSL'05*, LNCS 3634.
6. F. Blanqui. Rewriting modulo in Deduction modulo. *Proc. of RTA'03*, LNCS 2706.
7. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. of RTA'00*, LNCS 1833.
8. F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proc. of RTA'04*, LNCS 3091.
9. F. Blanqui. Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.
10. F. Blanqui, C. Kirchner, and C. Riba. On the confluence of λ-calculus with conditional rewriting. In *Proc. of FoSSaCS'06*, LNCS 3921.
11. W. N. Chin and S. C. Khoo. Calculating sized types. *Journal of Higher-Order and Symbolic Computation*, 14(2-3):261–300, 2001.
12. R. Davies and F. Pfenning. Intersection types and computational effects. In *Proc. of ICFP'00*, SIGPLAN Notices35(9).
13. M. Fischer and M. Rabin. Super-exponential complexity of presburger arithmetic. In *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*, 1974.
14. J. Gallier. On Girard's "Candidats de Réductibilité". In P.-G. Odifreddi, editor, *Logic and Computer Science*. North-Holland, 1990.
15. J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proc. of POPL'96*.
16. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems. *Theoretical Computer Science*, 121:279–308, 1993.
17. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2):3–29, 1998.
18. M. Presburger. ber die vollst ndigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Sprawozdanie z I Kongresu Matematykow Krajow Slowcanskich, Warszawa, Poland*, 1929.
19. W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the 1972 Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, 1975.
20. V. van Oostrom and F. van Raamsdonk. Comparing Combinatory Reduction Systems and Higher-order Rewrite Systems. In *Proc. of HOA'93*, LNCS 816.
21. H. Xi. *Dependent types in practical programming*. PhD thesis, Carnegie-Mellon, Pittsburgh, United States, 1998.
22. H. Xi. Dependent types for program termination verification. *Journal of Higher-Order and Symbolic Computation*, 15(1):91–131, 2002.